# HLA Data Distribution Management:
# Design Document Version 0.7

## Forward

This document represents the results of more than a year's effort by a team of individuals from several organizations, all focused on defining the initial description of Data Distribution Management as it will be implemented in multiple RTIs. The baseline DDM definition period has been characterized by in-depth and substantive debate about the goals of DDM, and the most effective balance between generality and performance. The DDM team was formed at DMSO's request and guided through it's initial stages by Steve Seidensticker. The team's members include Steve Seidensticker, Daniel Van Hook and James Calvin and their team at MIT-Lincoln Laboratories, Andreas Kemkes of Perceptronics, Mikel Petty of IST, Richard Weatherly of MITRE, Jeffrey Steinman at Metron, Richard Fujimoto of Georgia Tech, Reed Little from CMU's Software Engineering Institute, Larry Mellon and Darrin West from SAIC, and myself. The results of the team's efforts can be seen in the HLA Interface Specification, for which Reed Little has responsibility, the DDM implementations developed by the teams from MIT-Lincoln Laboratories and MITRE, and this Design Document for which I have assumed responsibility from Jeff Steinman. This document will evolve and companion documents will be developed as I guide the DDM team through subsequent phases of DDM development and experimentation.

Katherine L. Morse
Senior Computer Scientist
Science Applications International Corporation

## 1. Introduction

The purpose of this technical report is to document the functionality of the High Level Architecture (HLA) Data Distribution Management (DDM) services. This report is written primarily for those who are interested in how the DDM services work. While we have tried to explain how the Data Distribution Management services can be used, this document is really not intended to function as a user's guide. Rather, it is a design document intended to provide the technical background for the data distribution services in HLA. Subsequent to early experiments with the current RTI, a use case study with lessons learned will be produced which will serve as a user's guide. It will contain specific scenarios, code examples, and metrics from specific experiments. It is anticipated that these two documents and any accompanying documents identified as necessary later will be "living" documents that evolve as new DDM services are considered in HLA.

The design and use approaches described in this document rely solely on the DDM services specified in the HLA Interface Specification. So, everything presented here is equally relevant to any RTI implementation. Details of specific RTI implementations are provided at the end.

Currently this document addresses real-time Data Distribution Management. DDM in logical-time federates is still under investigation.

Section 2 briefly describes the HLA goals for Data Distribution Management, the interaction of DDM services with other HLA services, and aspects of federation DDM use

planning. Section 3 documents the multidimensional routing space approach. Section 4 then shows how to use the Data Distribution Management services in a physically correct manner, taking into account (1) the fact that subscription regions and update regions are sampled discretely over time (not continuously), and (2) the effects of latency. Section 5 provides detailed design information for specific Run-Time Infrastructure (RTI) implementations. Section 6 is a short bibliography of useful documents that provide background to the filtering topic described in this report.

## 2. Goals

The goal of HLA DDM services is to limit the messages received by federates in large distributed federations to those messages of interest in order to reduce (1) the data set required to be processed by the receiving federate and (2) the message traffic over the network. This functionality should be provided in a manner that is straightforward and easy to use. Thus, HLA Data Distribution Management services are concerned with three things:

- **Efficiency:** Data distribution management services should involve minimum overheads in terms of *computations*, *message latencies*, and *memory usage* for all of its services. Expensive operations, such as string comparisons, complex or costly computations, excessive handshaking, etc., should be avoided whenever possible. A service must justify the overhead cost it incurs. An expensive service may be acceptable if it saves more than it costs.
- **Scalability:** Data distribution management services provided by the RTI should scale in terms of (1) *computational complexity* for handling requests, (2) *message traffic* and/or bandwidth for distributing information, and (3) *memory requirements* for storing attribute information, maintaining tables, etc. It is understood that the services are not required to scale better than the physical problem[1]. The parameters that normally affect scalability are: (a) the *number of federates* (or hosts) in the federation, (b) the *number of simulated entities* per federate, (c) the average complexity concerning the *interests of each entity* (i.e., an entity may have a number of different kinds of sensors), (d) the *interaction rates between federates* after one discovers relevant objects in the other, (e) the *locality of objects*[2], and indirectly (f) the *scenario*.
- **Interfaces:** Data distribution management services must support the *right interfaces* to provide the *right filtering functionality* that is needed in HLA federations. The interfaces should be provided in an easy-to-use manner. It is important to capture the right functionality in the HLA Data Distribution Management services and to know where to draw the line between the RTI and software provided by the federates, which

---

[1] For example, if all of the simulated entities move to the same area of the battlefield and can therefore detect one another, then DDM cannot be expected to eliminate the $n^2$ nature of the physical problem.

[2] Objects that are local (i.e., within the same federate) can interact without requiring messages to be sent through the network. In large applications, it is sometimes possible to decide which federates create which objects. It often makes sense for objects that stay within the same physical region during the course of the simulation execution to be created within the same federate to provide better locality for the federation.

could be used to further automate much of the filtering for a federate. It is recognized that interfaces which describe limited functionality are usually easier to use than interfaces that describe complex and powerful functionality. The goal is to make the interfaces as easy to use as possible while supporting the essential HLA Data Distribution Management services. The interfaces must also be sufficiently general that different underlying implementations may support the same common interfaces.

# 3. Data Distribution Management in HLA

The Declaration Management (DM) services allow federates to limit the attribute update and interaction data they receive. A federate which subscribes to an object class' attribute values (or to an interaction class) will receive all updates of the specified attribute values for all objects of that class which currently exist in the entire federation. This type of filtering has the benefit of eliminating delivery of attributes for whole classes of objects which are of no interest to the subscribing federate. We'll refer to this as *class-based* filtering. For a small federation or one with few objects created in each class, class-based filtering may be sufficient to support performance and scalability requirements. However, for many large federations with large numbers of objects in their classes more detailed filtering may be required to improve performance and scalability. The DDM services provide *value-based* filtering.

The DDM services allow a federate to receive the subscribed attributes selectively based on values or characteristics of the publishing federate. For example, using only DM services, a federate could subscribe to attributes of all fixed wing aircraft. However, if the "playbox" is large, not all fixed wing aircraft may be in sensor range of the subscribing federate. In this case, the federate could invoke DDM services to limit the aircraft whose attributes it receives to those within its sensor range.

## 3.1 A Conceptual Overview of Routing Spaces

The fundamental Data Distribution Management construct is a *routing space*. A routing space is a multidimensional coordinate system through which federates either express an interest in receiving data or declare their intention to send data. These intentions are expressed through:

- *Subscription Region*: Bounding routing space coordinates that narrow the scope of interest of the subscribing federate[3].
- *Update Region*: Bounding routing space coordinates which are guaranteed to enclose an object's location in the routing space.

Both subscription and update regions can change in size and location over time as a federate's interests change or an object's location in the routing space changes.

[3] Regions in a multidimensional routing space do not necessarily map to physical geographical regions. A region in a routing space should be thought of as an abstract volume with any number of dimensions, e.g. radio channels.

*An object is discovered by a federate when at least one of the object's attributes comes into scope for the federate, i.e. if an only if:*

- *the object's attribute is not owned by the federate*
- *the federate has subscribed to the attribute*
- *the federate that owns the attribute is publishing the attribute, and*
- *the object's update region overlaps the federate's subscription region.*

Initially we will confine our discussion of DDM services to their application to objects and attribute updates. Section 3.4 addresses the application of DDM services to interactions.

DDM services extend DM and Object Management services with region parameters so that federates can specify by object class and attribute name the types of data they will send or receive, while also narrowing the specific instances of data. Each federate decides which of the federation routing spaces are useful to them and defines the portions of those routing spaces that specify *regions*, or logical areas of interest particular to the federate, by putting bounds (*extents*) on the dimensions of the selected routing space.

Specifying a subscription region, the federate tells the RTI it is interested in data which fall within the extents of the region specified by that federate. Specifying an update region and associating that update region with a particular object instance is a contract from the federate to the RTI that the federate will ensure that the characteristics of the object instance which map to the dimensions of the routing space fall within the extents of the associated region at the time that the attribute update is issued. This implies that the federate is monitoring these added characteristics for each of the attributes owned by the federate. As the state of the objects change, the federate may need to either adjust the extents on the associated regions or change the association to another region.

Figure 1 shows one update region (U1) and two subscription regions (S1, S2) within a two dimensional routing space. In this example, U1 and S1 overlap so attribute updates from the object associated with U1 will be routed to the federate that created S1. In contrast U1 and S2 do not overlap so attributes will not be routed from the federate that created U1 to the federate that created S2.

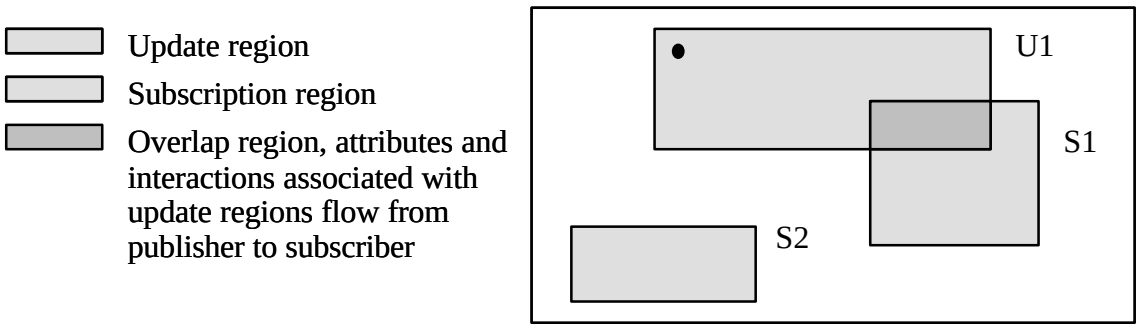



**Figure 1. Two-dimensional Routing Space Example**

When an update region and subscription region of different federates overlap, the RTI establishes communications connectivity between the publishing and subscribing federates.

The subscribing federates each receive only the object class attributes to which they subscribed, although they may receive individual updates outside their subscription region depending on the precision of the routing space implementation. In figure 1, S1's federate will receive attribute updates from the object associated with U1 because their regions overlap, even though the object itself is not within S1.

Each federate can create multiple update and subscription regions. Update regions are associated with individual objects that have been registered with the RTI. A federate might have a subscription region for each sensor system being simulated.

For the sake of simplicity we have described regions as n-dimensional rectangles up to this point. In fact, they are defined as sets of extents, or sets of n-dimensional rectangles. Regions which are not logically rectangular can be approximated by sets of smaller rectangles. Federates should carefully evaluate the performance impact of using such approximations against the cost of performing coarse filtering using rectangular regions and fine filtering in the federate.

## 3.2 Services Associated with Data Distribution Management

DDM services interoperate with HLA Object Management (OM) services and Declaration Management services as well as extending some of them in DDM-specific ways. OM services consist of the group of RTI services which deal with the creation, modification, and deletion of objects as well as their attribute updates. Specifically, OM, DM, and DDM services perform the following related functions[4]:

Object Management

- *Discover Object*
- *Update Attribute Values*
- *Reflect Attribute Values*

Declaration Management

- *Publish Object Class*

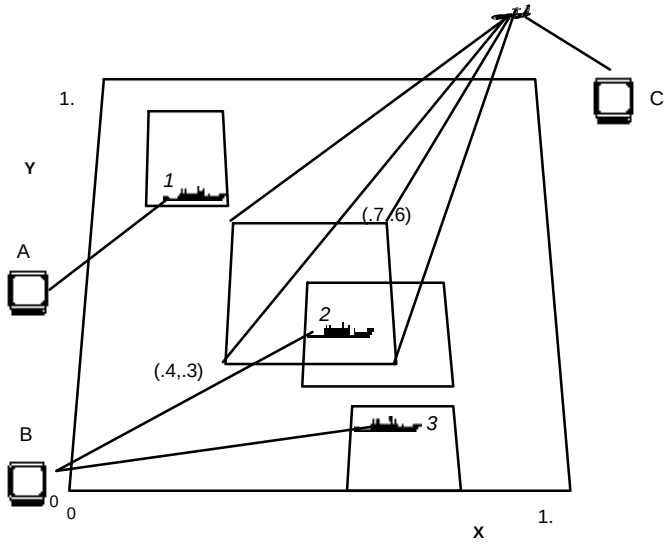
OM and DM services extended by Data Distribution Management

- *Register Object with Region* (OM)
- *Subscribe Object Class Attributes with Region* (DM)

## 3.3 Object Discovery and Attribute Reflection

An example will serve to illustrate the specific flow and interaction of DDM, OM, and DM services which allow a federate to discover objects of interest and to receive the objects' attribute values as they are updated. Figure 2 shows an example of routing space usage based on position.

[4] This is not a complete list, just the ones necessary to understand the fundamental flow of DDM. The remainder of DDM and DDM-related services are described in section 3.5.




1. The federation defines a 2D routing space with dimension variables X and Y, both with ranges [0,1]

2. Federate C subscribes for ships in the region specified to the RTI by the extents {{.4,.7},{.3,.6}}

3. Federates A and B publish ships and associate regions {{.2,.4},{.7,.9}}, {{.5,.8},{.3,.5}}, and {{.6,.8},{0,.2}} with instances 1, 2, and 3, respectively

4. The RTI tells federate C to discover ship 2

5. The RTI only reflects attributes for ship 2 to federate C - ships 1 and 3 are filtered

**Figure 2. Playbox routing space example.**

Routing Space Definition. The federation begins the process by defining a routing space with dimension variables in the Federation Execution Data (FED)[5] file. Several important quantities are specified in each routing space, including:

- Name of the routing space
- Number, names, and ranges of dimensions in the routing space
- Attributes to be routed through the routing space

The federates must agree on the use of each dimension, i.e. the routing variables on each axis, their range, and their interpretation[6]. Since this definition happens during the federation development process and its results are interpreted automatically by the RTI at federation initialization, there are no RTI service calls associated with this step. For our example we will assume that attributes location, velocity, and force affiliation may be routed through this routing space. Federates A and B are going to publish all three attributes, but Federate C is only interested in location and force affiliation.

Subscription. Federates create subscription regions to model their data requirements and/or sensors. In the example, federate C creates a subscription region for attributes location and force affiliation, and corresponding to the sensor range of the aircraft: *Create Region, Subscribe Object Class Attributes with Region.*

Publication. Federates express their intention to publish specific attributes of object classes to the RTI. In the example, federates A and B publish objects of class ship, and attributes location, velocity and force affiliation: *Publish Object Class.*

[5] See section 3.8 for a description of the Federation Data Distribution Model process by which a federation decides this information.

[6] Section 3.7 addresses these concepts in more detail.

Registration. Federates create update regions and objects, then register the objects and their regions with the RTI. As a result, the RTI associates IDs with the objects and may notify the creating federate that it owns all of the objects' attributes[7] that the federate registered. In the example, federate A's ship has ID 1; federate B's ships have IDs 2 and 3: *Create Region, Register Object with Region, Attribute Ownership Acquisition Notice*.

Discovery. The RTI determines which federates should discover which objects by matching subscription and update regions. Object discovery is provided only once by the RTI to a federate even when multiple locally-defined subscription regions overlap a remote object's update region. In addition, the RTI determines if any of the discovered objects' published attributes are of interest to the subscribing federates. Federate C is notified by the RTI to discover federate B's ship 2, and that ship 2's attributes location and force affiliation are in scope : *Discover Object, Attributes in Scope*.

Reflection. The publishing federate updates an object's attributes to the RTI. The RTI notifies all subscribing federates that they should reflect the updated attributes of interest. Even though federates A and B may update the attributes of ships 1, 2, and 3, federate C only receives the updated attributes, location and force affiliation, of ship 2 because it is the only ship object that federate C has "discovered". The updated attributes of ships 1 and 3 may be sent to other federates (not shown) that have discovered them via other subscription regions, or their attributes may not be sent anywhere if no other federate has subscribed to them: *Update Attribute Values, Reflect Attribute Values*.

### 3.4 DDM for Interactions

The fundamental difference between interactions and object is that objects persist, but interactions are instantaneous events. For this reason object discovery and attribute updates are separated, while interactions are "discovered" at the same time their parameters are received. With this exception, DDM services for interactions are analogous to DDM services for objects and attributes.

Routing Space Definition. The routing space definition process is the same with the exception that all of the parameters of an interaction class must be routed through a single routing space rather than split over multiple routing space as an object class' attributes may be.

Subscription. A subscribing federate may only subscribe to the interaction class as a whole, not to individual parameters: *Create Region, Subscribe Interaction Class with Region.*

Publication. Federates express their intention to publish interaction classes, once again the entire class rather than a subset of parameters: *Publish Interaction Class.*

Registration. Registration is not applicable to interactions because they don't persist.

[7] Attribute ownership notification may be optionally controllable by the RTI. See section 5.1 for a complete discussion of RTI DDM control switches.

Discovery. Discovery is not applicable to interactions because they don't persist, but the RTI will provide advisory notification to subscribing federates that they may receive interactions as a result of overlapping interaction class update and subscription regions just before the interaction is received: *Interaction in Scope*.

Reflection(Receive). The publishing federate sends interactions which are received by subscribing federates. Note that an individual interaction's "update region" is sent with the interaction, so that region intersections are calculated by the RTI just before the interaction is received by the subscribing federate. A publishing federate may choose to send only a subset of the interaction class' parameters: *Send Interaction with Region*, *Receive Interaction*.

## 3.5 Other DDM and DDM-Related Services

In addition to the services required to support basic DDM functionality as described in sections 3.3 and 3.4, there are several other DDM and DDM-related services.

### *3.5.1 DDM Services*

As an object's state variables or a federate's state change over time, it might become necessary to modify update or subscription regions. Note that this does not have to occur every time an attribute is updated. When an update region or a subscription region is modified, the changes are passed to the RTI using the *Modify Region* service. The RTI then reassesses the matches between the modified region and all regions of the complementary type.

It is possible for a federate to have an object stop updating or subscribing by making calls to *Delete Region*. These can be made for each region associated with an object so that the RTI knows not to continue sending or receiving information from other federates concerning a particular object.

A federate may undo its subscription to an object or interaction class through a region with the *Unsubscribe Object Class with Region* and *Unsubscribe Interaction Class with Region* services. A federate may dissolve the association between an object and an update region using *Unassociate Region for Updates* or may create a new one with *Associate Region for Updates*. The original association between an object and an update region is usually made using the *Register Object with Region* service.

Under some circumstances, a federate may wish to specifically request the values of attributes rather than waiting for objects to update them. For example, if a subscription region is changing frequently it may intersect an update region only briefly during a period of time when the object associated with the update region performs no attribute updates. However, the subscribing federate's correct behavior may depend upon knowing the attribute values of the object and acting on them. The subscribing federate may issue a *Request Attribute Value Update with Region* call to obtain the necessary value(s). Note that this circumstance can also arise if the update region is changing quickly. This scenario presents opportunities for switch-controlled RTI optimizations which are described in section 5.1.

In RTI implementations that use gridded DDM approaches, a region may overlap only a portion of a grid cell. However the entire grid cell is still effectively part of the region. In these circumstances it is advantageous for the RTI to notify the federate so that the federate may optimize its region changes, particularly if a region change won't actually change the grid cells the region would overlap. The *Change Thresholds* service callback provides the federate with this information.

### *3.5.2 Declaration Management Services*

While determining which objects and their attributes are in- and out-of-scope for other federates, the RTI may determine that there are no subscribers for whole object classes or some of their object instances. The RTI will notify the publishing federate not to send any updates until notified otherwise using the *Turn Updates Off for Object Class* or *Turn Updates Off for Object Instance* services. If the objects and/or attributes come back into scope later, the publishing federate will be notified to resume updates using the *Turn Updates On for Object Class* or *Turn Updates On for Object Instance* services. These same activities apply to interactions, but only at the level of classes. The interaction control services are *Turn Interactions On* and *Turn Interactions Off*.

*Unpublish Object Class* and *Unpublish Interaction Class* have the same results under DDM that they do under DM: objects of the unpublished class will be removed and their attributes will go out of scope; subscribers to the interaction class will be notified that the class is now out of scope.

### *3.5.3 Object Management Services*

When a federate uses the *Delete Object* service, the RTI automatically removes its associated update regions from its internal structures and notifies federates reflecting the object's attributes with a *Remove Object* callback.

When a federate's subscription region no longer intersects the update region of an object, the RTI services must coordinate actions so that the receiving federate knows to delete its local representation of the object (assuming that no other locally defined subscription regions in any of the routing spaces intersect the update region of the object). The RTI tells the subscribing federate to remove the object's representation[8] via the *Remove Object* callback. *Remove Object* may also be triggered by the publishing federate invoking *Delete Object*.

It may also be the case that the federate should continue to know about an object, but some of its attributes may not be up to date. This situation can arise when a federate subscribes to attributes of an object class through different routing spaces. If the federate's subscription region in one routing space no longer intersects the object's update

[8] The HLA does not specify how federates internally represent remote objects which are discovered via the RTI. The HLA only specifies that federates receive *Discover Object* and *Remove Object* messages and behave consistently internally.

region in that routing space, the attributes routed through that space go out of scope. The federate may receive an *Attributes Out of Scope* callback[9].

### 3.6 DM-DDM Interoperability and Default Regions

Under some circumstances, federations may want to include federates which only use DM for filtering with federates that use DDM. The HLA definition guarantees correct and consistent semantics for both types of federates in the same federation, but federations should carefully consider the performance impact of a DM-only federate receiving large quantities of data after the federation has constructed routing spaces to reduce data traffic on the network.

All attributes are always associated with a routing space and a region, either implicitly or explicitly. If an attribute is not explicitly associated with a routing space in the FED, it is implicitly associated with the *default routing space*. Even when an attribute is associated with a routing space in the FED, if an individual object instance is registered with the RTI without a region, all of the object's attributes are associated with the *default region* in the routing space. The default region's extents are the maximum bounds of the dimensions of the routing space. A subscribing federate can only subscribe to an attribute through the routing space with which the attribute is associated. If no region (and consequently, no routing space) is used in a subscription call, i.e. with *Subscribe Object Class Attributes*, the RTI subscribes to the default region in the correct routing space: either the attribute's routing space or the default routing space if the attribute has none. Unsubscribe services work on a per-region basis, i.e. *Subscribe Object Class Attributes with Region* is only negated by a call to *Unsubscribe Object Class with Region* using the same region designator. By extension using the default routing space concept, *Subscribe Object Class Attributes* is only negated by a call to *Unsubscribe Object Class*, both with no region.

The following example illustrates the interaction of multiple subscription calls within a single federate. In the example RS is the routing space through which all of object class A's attributes {x, y, z} are routed.

1. R1 = Create Region (RS, extent_set1)
2. Subscribe Object Class Attributes with Region(A, {x, y}, R1)
3. R2 = Create Region (RS, extent_set2)
4. Subscribe Object Class Attributes with Region(A, {x, z}, R2)
5. Subscribe Object Class (A, {y, z})

The federate can now be assured of getting all updates of attributes y and z from all instances of class A because the federate has subscribed to the default region of RS by using *Subscribe Object Class*. The federate may or may not get updates of attribute x depending on the overlap of R1 and R2 with update regions of instances of A. If the federate subsequently calls *Unsubscribe Object Class* on A, it is no longer assured of

[9] See section 5.1 for a description of an optional RTI switch to control this callback.

getting updates of y and z, but may get them if R1 and R2 overlap update regions appropriately, just as has been the case for attribute x all along. If the federate then calls *Unsubscribe Object Class with Region* on A with region R1, it will no longer get any updates of y, but its subscription through R2 may cause some updates of x to be delivered to it. The federate's status with respect to updates of z is unchanged.

From this example we can extrapolate how a DM-only federate can operate in a federation with DDM-using federates. When the DM-only federate subscribes to an object class, the RTI automatically uses the default region in the correct routing space, either the routing space through which the attributes are routed or the default routing space. All attribute updates for the subscribed class have to go through this routing space, by definition, and the DM-only federate gets them. When the DM-only federate publishes an object class and sends attribute updates, the RTI automatically routes them through the default region in the correct routing space and any subscribing DDM-using federates will get them since the default region overlaps all other regions of a routing space, also by definition. Notice that the DM-only federate observes exactly the same kind of behavior it would expect in a federation when all federates are only using DM. Default routing spaces and regions apply analogously to interactions without the complexity of attributes.

## 3.7 Routing Space Dimensions and Normalization Functions

The dimensions of routing spaces need not necessarily map to object attributes or interaction parameters in the Federation Object Model (FOM). The Data Distribution Management services provide a federation the capability to specify data distribution on data other than that exchanged as part of federation execution so long as all the participating federates agree on the meaning of the dimensions and have the data to calculate coordinates in the dimensions. However, by specifying routing spaces dimensions as attributes specified in the FOM, a federation can achieve strict value-based filtering.

The RTI does not understand the semantics of routing spaces or their dimensions, including type, range, and scale. So every RTI has a fixed minimum and maximum value for all dimensions of all routing spaces; we'll refer to them as RTI_min and RTI_max. The federates themselves must ensure that they map region extents to RTI_min and RTI_max using normalization functions. Obviously federations must agree about the normalization functions they will use for each dimension of each routing space or the federation will not exhibit correct semantics. A more detailed treatment of normalization functions is given in the Object Model Template (OMT) Extension for the Data Distribution Model Template.

## 3.8 The Federation Data Distribution Model

To use routing spaces, each federation defines the allowable routing spaces for the federation execution, including the dimensions of the routing space. The routing space dimensions are coordinated manually at federation planning time. Routing spaces are specified in the FED file with a name, the number of dimensions, and additional parameters. Object attributes are assigned to routing spaces in the FED file as well. The

OMT Extension for the Data Distribution Model Template describes the format and contents of a Federation Data Distribution Model (FDDM). An FDDM is analogous to a FOM; it records federation-global agreements, but with respect to the use of DDM rather than object and interaction data. The FDDM records both data that is translated into the FED for use by the RTI, such as routing space names and dimensions, and attribute and interaction assignments to routing spaces, as well as information to be used by the federates to ensure correct use of DDM services such as the agreed-upon type, range, and normalization functions of routing space dimensions. It is the former type of data which the RTI uses to enforce the correct routing of attributes and interactions through the expected routing spaces.

# 4. Physically Correct Filter Strategies

Network latencies and lookahead restrict how tightly in time one federate can interact with another federate, for example, to request object attributes. In addition to network overheads, there are other factors that affect filtering. Some routing space coordinates used to describe update regions (motion, for example) change in a continuous manner over time. These are called *continuous coordinates*. Filtering strategies must somehow sample these continuous coordinates at rates that are not too high. Otherwise, the filter strategy might require too many computations and changes in the filter specifications. However, if the routing space coordinates are not sampled often enough, the filter will not be as responsive or as efficient as it otherwise could be.

Similarly, subscription regions can also change continuously over time. An efficient filtering strategy must not sample subscription regions too often or too infrequently for the same reasons. The filter strategy used by a federation must take into account the fact that true update regions and true subscription regions may not be accurately represented by the sampled regions that are represented as static regions at discrete points in time. Two ways of getting around this problem are (1) to widen subscription regions, and/or (2) to widen update regions in a physically correct manner. This must be done in a way that preserves true overlaps between the update and subscription regions.

There are other kinds of routing space coordinates that change their values in unpredictable manners. These are called *discrete coordinates*. Here, the routing space coordinate values remain constant unless they are changed to new values at discrete points in time.

## 4.1 Filtering on Discrete Coordinates

Discrete coordinates can be thought of as step functions[10]. Their true physical value remains constant until changed. Discrete coordinates are generally not continuous variables but actually make large jumps when they change. If changes to discrete coordinates can be predicted, then both the update regions and subscription regions can take advantage of this fact to provide physically correct filtering by (1) adding a new extent early that indicates the predicted change and then (2) removing the old extent when

[10] Enumerated types are the most obvious example of this type.

it is no longer valid (i.e., when the discrete coordinate actually changes its value). If it is not possible to predict when discrete coordinates change their values, then the filtering will still be logically correct, but not physically correct, i.e. objects might be discovered late but it will be done in a repeatable, well-defined manner when logical time management services are used.

Filtering on discrete coordinates can be very useful in describing categories of objects to which a federate might want to subscribe. For example, a federate might want to subscribe to objects that are radar-detectable. This could be one of the dimensions in a routing space. Radar detectability could be a dimension with two bins (True or False) which are set during initialization and never change. This kind of coordinate dimension can be useful in specifying the static characteristics of objects beyond just their class type, which is the most basic type of filtering that is provided by the Declaration Management services.

## 4.2 Filtering on Continuous Coordinates

We differentiate continuous coordinates from discrete coordinates in the sense that they can be described as continuous functions over time. In other words, they are not represented as step functions, but instead as continuous curves that smoothly change over time. We further assume that these curves can be evolved over time in software according to known equations, integrals, etc.

The RTI does not automate filtering based on known functions. It is the responsibility of the federates to change routing space regions when coordinate values in the routing space change appreciably. Future Data Distribution Management services could automate many of the filter set-up tasks for continuous coordinates that have known functions (e.g., standard motion types, user defined functions, etc.) but this topic is beyond the scope of this discussion[11].

We assume that at any point in time, the equations can change form or characteristic parameters. However, we assume that it is possible to bound how much a routing space coordinate value can change during a specified time. In other words, if a routing space coordinate has a value $v$ at time $t$, then at most it can change by $dv$ at time $t+dt$. By bounding how much a continuous coordinate can vary over time, it is possible to establish efficient filtering practices that also provide physically correct filtering. To define filter specifications correctly with latency, it may also be important to bound the worst case $dv$ for any object in the federation (which is discussed later). This has to be done for each continuous coordinate in the routing space.

The goal of this section is to discuss how much to artificially widen the subscription regions of continuous coordinate values and how often to sample the interest regions and the related update regions. It is important to recognize that in practice, one would never want to sample an object's routing space coordinates independently, but rather, their values would always be sampled together in a coordinated manner. Nonetheless, our current discussion focuses on a single routing space coordinate and how to determine its natural sampling rate. Federates need to take all of their routing space dimensions into account when determining their sampling rates. Federates may wish to further coordinate this process so that all of their local objects modify their update regions together with subscription regions.

[11] This is probably a good role for middleware.

We start this discussion by first neglecting latency and consider only the sampling of update regions and subscription regions. The key to understanding how this works is to recognize that subscription regions must be artificially expanded to always include their true region of interest within their sampling period. Similarly, update regions can (1) be widened, or (2) the subscription regions of other federates must be further expanded to account for the fact that update regions are also sampled.

### *4.2.1 Dynamically Changing Subscription Regions*

First consider the sampling effects of subscription regions. Suppose that a federate's true subscription region is specified by a low and high value, $[v_{lo}, v_{hi}]$. However, when sampling the subscription region $\Delta t$ time units later, the old values might have changed by as much as $\Delta v$. In order to keep the specified regions valid until the next sampling time, we must extend the subscription region by $\Delta v$ on both ends[12]. In other words, the extents must be defined as $[v_{lo}-\Delta v, v_{hi}+\Delta v]$.

It is important to recognize that we could either (1) specify a value for $\Delta v$ which then dictates our choice for $\Delta t$, or we could (2) specify a value for $\Delta t$ which then dictates our choice for $\Delta v$.

*What is the best way to define $\Delta v$ and $\Delta t$?*

Probably, the best way to do this is to first make sure that $\Delta t$ is not too small. Otherwise federates will be required to redefine their subscription regions too frequently to be practical. Therefore, we should first define $\Delta t_{min}$ as the minimum time between filter updates that makes sense. It is possible for the expanded subscription region to be much larger than the true region of interest. This is the tradeoff that one has to make concerning filter overheads and the benefits of filtering itself. Corresponding to $\Delta t_{min}$ is $\Delta v_{min}$. Once $\Delta v_{min}$ is determined, we might find out that $\Delta v_{min}$ is still much smaller than what makes sense. For example, if routing space bins are used, and their resolutions are much larger than $\Delta v_{min}$, then without losing much efficiency (maybe none at all), we should be able to extend it further to $\Delta v_{bin}$, where $\Delta v_{bin}$ is related to the bin size (probably a reasonable fraction of a bin) for that routing space dimension. Remember, the subscription regions include any cells that they might overlap in the routing space so it does not make sense to define subscription regions that are much smaller than bin sizes. In this case, the corresponding $\Delta t$ should be set to $\Delta t_{bin}$.

### *4.2.2 Dynamically Changing Update Regions*

Update regions are also sampled. It is important not to sample an update region too often in order to keep filtering efficient, but on the other hand, not too infrequently so as to make the filter ineffective. Keep in mind that it is possible for an object's routing space position to be just outside a federate's subscription region, but then as its true physical state changes, it actually moves into the federate's subscription region before the next sampling period. To account for this, either the object's position has to be represented as an extended region, or the subscription regions from other federates have to be further widened.

We have the same problem as when sampling subscription regions (i.e., how often do we sample?). Assume that the maximum amount the coordinate value can change in time $dt$ is $dv$. The arguments are essentially the same as given in the subscription region sampling discussion if update regions are represented as extended regions. However, if

[12] It is possible to define a different $\Delta v$ for the low and high ends of a filter specification but for simplicity, we have assumed that they are the same.

subscription regions are expanded to account for the sampling of unextended update regions, then the subscription regions must be further expanded according to some known amount. This limits the choices on how to determine *dv* and *dt*. As with subscription regions, the most obvious approach is to choose *dt* to be not too frequent, and calculate *dv* based on *dt*. If *dt* is too small, then it might make sense to make it larger and extend the update regions in order to make up the difference.

### *4.2.3 Filtering Efficiency*

The filtering efficiency of update and subscription regions can be characterized based on their extensions. For simplicity we assume a two-dimensional, geographic region with dimensions x and y. Figure 3 illustrates a more efficient subscription region and a less efficient subscription region. Even though $S_1$ is larger than $S_2$ in absolute terms, it is more efficient than $S_2$ because it is extended less proportionally. Conversely, if $\Delta x_1$ and $\Delta y_1$ were proportionally larger than $\Delta x_2$ and $\Delta y_2$, $S_2$ would be more efficient than $S_1$.

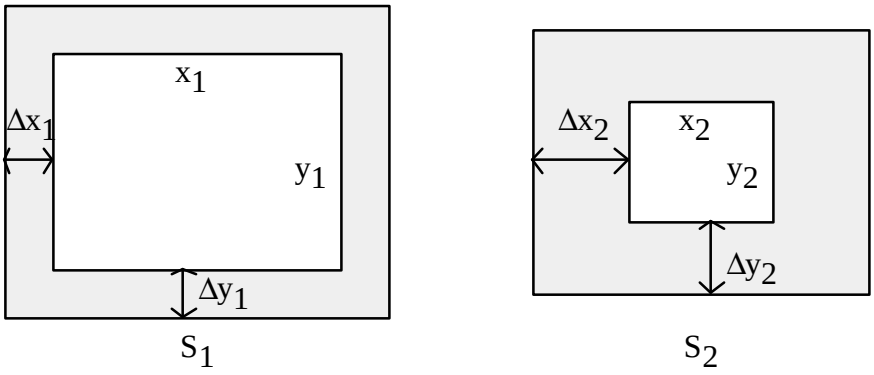



**Figure 3. Characterizing the efficiency of subscription regions**

Intuitively, $S_1$ is more efficient than $S_2$ because it will tend to get fewer "false positive" updates: updates which fall into the extended region, but not in the actual region. A subscription region's efficiency is:

$$\eta = xy/[(x + 2\Delta x)(y + 2\Delta y)] \quad (1)$$

The smaller the extension of the region is, the closer $2\Delta x$ and $2\Delta y$ are to zero, and the closer the region's efficiency is to one. Notice that equation (1) does not say anything about the absolute size of the subscription region.

In general, this same characterization of efficiency cannot be applied to update regions because most entities are logically single points[13]. Under such circumstances, the update region is all extension. Reflecting on our subscription region example, x and y would both be zero, resulting in the region's efficiency being zero. This is obviously not a useful measure. As we'll see in the next derivation, an update region's effect on filtering efficiency is related to the size of the subscription with which it intersects. Consider the intersection of subscription region $S_1$ with update region $U_1$ in Figure 4.

[13] Obviously this is not the case for large entities such as weather effects, but the derivations for efficiency apply equally.

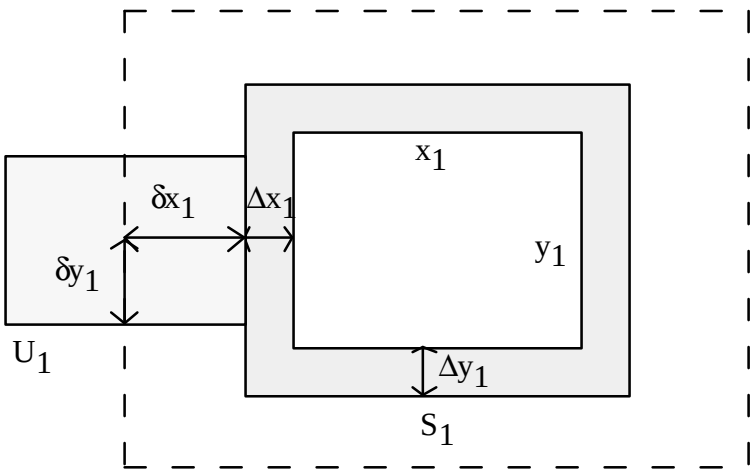



**Figure 4. Filtering efficiency impact of subscription and update region intersection**

The entity in $U_1$ is assumed to be a point in the center of the update region. If we were to extend $S_1$ rather than $U_1$ to account for the entity's rate of change and still maintain physically correct filtering, $S_1$'s dimensions would have to be $x_1 + 2\Delta x_1 + 2\delta x_1$ by $y_1 + 2\Delta y_1 + 2\delta y_1$. The result would be yet more "false positive" updates for $S_1$. Plugging these extents back into our original equation for efficiency, we have that the efficiency of the filter created by the intersection of an update and subscription region is

$$\eta = xy/[(x + 2\Delta x + 2\delta x)(y + 2\Delta y + 2\delta y)] \quad (2)$$

If $\delta x$ and $\delta y$ are large relative to x, y, $\Delta x$ and $\Delta y$ the filtering efficiency will be low. If $\delta x$ and $\delta y$ are small relative to x, y, $\Delta x$ and $\Delta y$ the filtering efficiency will less impacted. So, it's not the absolute size of the update region that we use to categorize them, but their size relative to subscription regions with which they intersect. Even if an update region has a large absolute size, if all the subscription regions in the federation are several times as large, the update region's size will have little impact on filtering efficiency. Likewise, even an update region with small absolute size may have a large impact on efficiency if all the subscription regions are also small.

### *4.2.4 Accounting for Latencies*

The discovery process takes a certain amount of time to occur. During the time $L_{tot}$ (where $L_{tot}$ is the total latency required for object discovery)[14], continuous coordinates may vary enough to warrant object discovery. To accommodate this fact, it is necessary for update and subscription regions to once again be artificially widened to account for the dynamic changes possible between an object's subscription region and any other object's update region. Assume that the value *e* corresponds to the maximum relative change for a coordinate value *v* corresponding to a given latency $L_{tot}$ (required for object discovery). Then, update and subscription regions have to be widened by the amount *e* in order for them to be physically correct, i.e. the filter does not miss any objects that it should discover over time. If update regions are represented as sampled points in the routing

[14] The total latency will be dependent on the algorithm used for coordinating the filtering. In real-time federates, $L_{tot}$ involves the latencies for discovering objects. If heartbeat messages are used without push and pull handshaking for object discovery, then $L_{tot}$ will also include the time between heartbeats.

space, the subscription regions will have to be further widened to account for the maximum *v* in the federation.

### *4.2.5 Multiple Continuous Coordinates*

We have discussed four important aspects of physically correct filtering on a continuous coordinate value. In practice, a routing space may have many continuous coordinates in its definition. Think of these continuous coordinates as a vector, $\vec{v}$. It is important to remember that updating routing space regions for these coordinate values should be done together instead of having each dimension schedule its own updates independently. Therefore, subscription regions should be sampled every $\Delta t$ time units and update regions should be sampled every $dt$ time units[15]. Because of the finite sampling rates, and due to network latencies, regions must be increased accordingly. This is shown in Table 1.

**Table 1. Time values and coordinate increases.**

| | Time Values | Coordinate Increase |
|---|---|---|
| Subscription Region | $\Delta t$ | $\Delta \vec{v}$ |
| Update Region | $dt$ | $d\vec{v}$ |
| Latency | $L_{tot}$ | $\vec{e}$ |

If update regions are represented as sampled points in a routing space, the subscription regions are expanded as follows:

$$[\ \vec{v}_{lo} - \Delta\vec{v} - d\vec{v} - \vec{e}\ ,\ \vec{v}_{hi} + \Delta\vec{v} + d\vec{v} + \vec{e}\ ]$$

Of course, care must be taken in the above expression to ensure that the specified regions do not exceed the minimum and maximum bounds for any of the routing space dimensions[16].

## 5. DDM Implementations

This section is designed to provide information about specific implementations of the DDM and useful insights into their use.

### 5.1 RTI Switches

Recognizing that the scope of DDM use can vary widely across federations, several optimizations are possible which can be controlled through switches in individual RTIs. With the exception of the Automatic Attribute Update switch, the switches described are

[15] There is no reason for subscription region sampling rates and update region sampling rates to be the same. In fact, it is possible for each object to define its own sampling rates based on their characteristics. For example, slow moving objects may not have to sample as often as fast movers.

[16] The choice of whether to expand update regions or just subscription regions is highly dependent on several factors including the implementation of DDM, network latencies, lookahead, and relative rates of change of update and subscription regions in the federation.

optional. The required default setting is given for each. If an RTI does not implement a switch, it must exhibit the default setting behavior.

Most of the switch suggestions are motivated by the "fly-by region" situation. Either an update or subscription region is moving so quickly that the two may only intersect briefly during a period of time when the object associated with the update region makes no updates. The subscribing federate may or may not want to know about the existence and state of the object. The switches are designed to allow federations to optimize the performance of the RTI to suit their needs.

### *5.1.1 Automatic Attribute Update*

Default behavior: when an attribute comes into scope as a result of a new intersection of the object's update region and a subscription region, the RTI automatically issues a *Request Attribute Value Update*.

Switch: the RTI does not issue a *Request Attribute Value Update*, but leaves that decision to the subscribing federate.

Required: yes.

### *5.1.2 Just-in-Time Discovery*

Default behavior: when an attribute comes into scope as a result of a new intersection of the object's update region and a subscription region, the RTI notifies the subscribing federate with *Discover Object*.

Switch: the RTI does not issue a *Discover Object* unless the object performs *Update Attribute Values* on the subscribed attribute(s), in which case the RTI issues a *Discover Object* callback immediately preceding the corresponding *Reflect Attribute Values*.

Required: no.

### *5.1.3 AOS/AIS Enable*

Default behavior: when an attribute comes into scope as a result of a new intersection of the object's update region and a subscription region, the RTI issues an *Attribute In Scope* callback to the subscribing federate. Analogously, the RTI issues an *Attribute Out of Scope* callback when the update and subscription regions no longer intersect.

Switch: the RTI does not issue either *Attribute In Scope* or *Attribute Out of Scope* callbacks and the subscribing federates simply get *Discover Object* and *Reflect Attribute Values* callbacks.

Required: no.

### *5.1.4 Ownership Notification*

Default behavior: when a federate registers a new object the RTI will respond with an *Attribute Ownership Acquisition Notification* callback for all attributes of the object.

Switch: the RTI does not issue an *Attribute Ownership Acquisition Notification* callback. The federate is expected to understand implicitly that it has ownership of all attributes.

Required: no.

## 5.2 Initial Implementation

The initial implementation of software to support DDM services was developed in cooperation with DARPA and the Synthetic Theater of War (STOW) Advanced Concept Technology Demonstration. This implementation served as the proof-of-concept prototype for DDM services in the government-provided RTI software. Further description of this implementation is available in published papers by Van Hook, Calvin et al (see bibliography).

This implementation relies on multicast groups and TCP connections to establish communications connectivity between subscribing and publishing federates. It is important to note that it separates the process of establishing connectivity between publishers and subscribers from the process of sending data. Once the RTI finds a match between update and subscription regions and makes the connection, updates are sent directly. This separation improves performance because the RTI does not have to check every attribute against every subscription region. However, it also means that filtering is not perfect because it is at the level of regions. In the example in Figure 1, all updates in region U1 would flow to the federate that created S1, not just the updates in the portion where they overlap.

This implementation of the HLA/RTI DDM services uses a gridded approach to perform these operations. A grid is used to efficiently determine which subscription and update regions overlap as well as to efficiently calculate and setup the required network connectivity to permit data to be routed from publishers to subscribers.

Each routing space defined by a federation is partitioned into a grid of cells. The parameters of the grid for each routing space, such as cell density, are defined in a file that is read in upon initialization. All federates' RTI interfaces use the same grid parameters. The number of dimensions of each grid cell is the same as the number of dimensions in the routing space, e.g. a three dimensional routing space uses three dimensional cells.

Associated with each grid cell is a parameter called a "stream". A stream can be thought of as a logical multicast group, i.e., it defines a collection of federates that receive any data sent to the stream. A stream does not imply any particular underlying transport service such as IP multicast, TCP connections, MPI, shared memory queues, etc. It only defines groups of receivers. Streams are algorithmically assigned to cells and all federates' RTI interfaces use identical mappings of streams to cells.

Comparison of subscription regions with update regions to determine overlap for purposes of establishing connectivity is performed implicitly using the grid defined on each routing space. No explicit comparisons are performed between subscription and update region extents. Instead, each federates' RTI interface maps its regions onto a routing space grid and determines a set of streams from the cells that fall within each region. If a cell

falls within a region, either partially or wholly, then that cell's stream is added to the set. Cells that lie completely outside a region are ignored.

Streams determined from mapping subscription regions to a routing space grid are joined by that federate's RTI interface. Streams determined from mapping update regions to a routing space grid are used to send attributes and interactions by that federate's RTI interface. The RTI sets up the necessary connectivity to permit data sent to a particular stream to be delivered to all federate's RTI interfaces that have joined that same stream. The implementation of this connectivity will be different depending on the actual transport service selected by the federate for each attribute or interaction class to be distributed, i.e. IP multicast groups, TCP connections, etc.

In the current design, published object-class attributes are identified with at most one routing space. Published and subscribed attributes should not be confused with the meaning of the coordinates in a routing space. Two different routing spaces, for example, can include position dimensions (along with other relevant dimensions) in their definitions. However, the two routing spaces must publish and subscribe to different attribute values. Object attributes are not allowed to span more than one routing space. This is shown in Figure 5. If an attribute were allowed to be associated with multiple routing spaces, then whenever the attribute is updated, a message for each multicast group in each routing space would have to be sent. Subscribing hosts will receive the message several times when their subscription regions in different routing spaces overlap the update regions. All of this could be wasteful. This is why the current approach only allows an attribute to be associated with a single routing space.

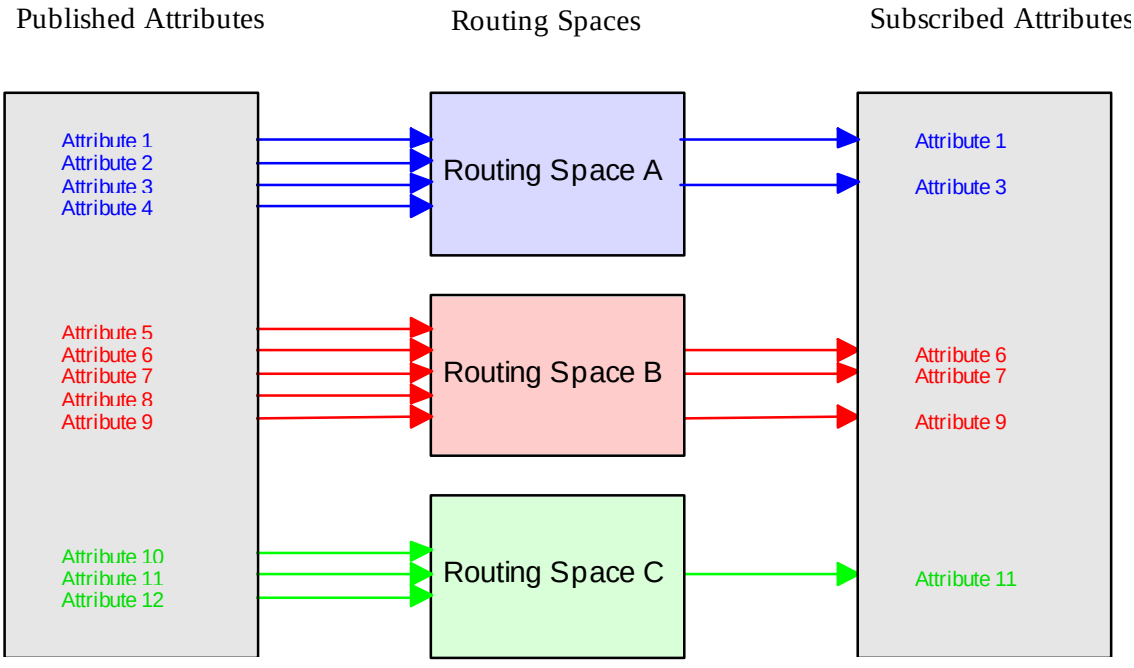
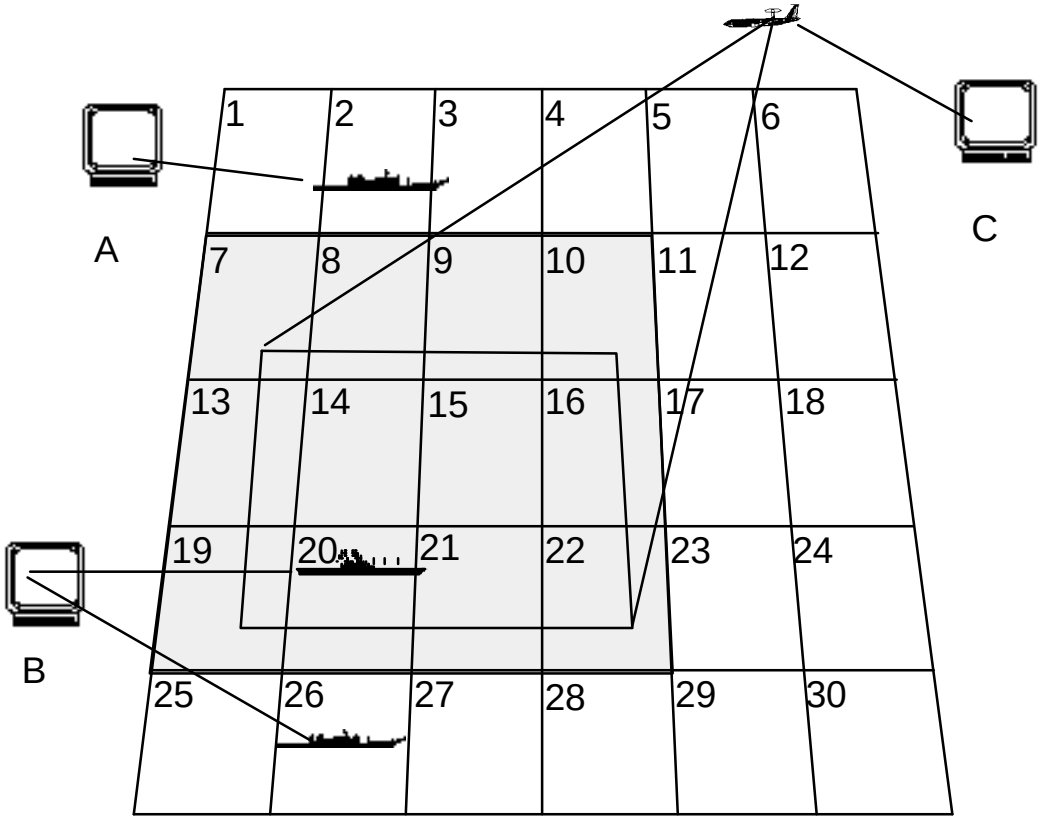



**Figure 5. Mapping of object attributes to routing spaces. Publishing objects map their attributes to a routing space (an attribute cannot belong to more than one routing space). Subscribing federates do not have to subscribe to all of the published attributes in a routing space.**

Each bin-cell maps to a stream which may be multiplexed with several other streams over one IP multicast group or to one or more TCP connections[17]. An object participating in the routing space normally defines its update region in one and only one bin at a time[18]. However, the subscribed interest regions of federates usually encompass several bin-cells simultaneously. When a federate's subscription region overlaps a set of bin-cells, all remote objects that are simulated by another federate and are updating in those bin-cells should be discovered by the federate. Figure 6 maps the example provided in Figure 2 to this binning approach using streams. Federate C joins the streams for the cells overlapped by the aircraft's subscription region. Updates for the ships simulated by federates A and B are sent to the streams associated with the cells that overlap each of the respective ships' update regions.




Simulation C joins 7,8,9,10,13,14,15,16,19,20,21,22
Simulation A sends to 2
Simulation B sends to 20 and 26
C receives updates sent to 20, but NOT 2 and 26

[17] The current routing space implementation uses fixed-sized bin-cells to map update and subscription regions into multicast group addresses. The choice of bin size in each dimension can strongly affect performance. The federates are able to define their bin sizes through the FED. Future Data Distribution Management services may not be limited to fixed-sized bins.

[18] This can be extended if necessary, e.g., weather objects such as clouds could extend over several bin-cells, aggregated sets of objects could define multiple points in the routing space which span several bin-cells.

**Figure 6. Mapping of update and subscription regions to streams.**

The current scheme using fixed and equal sized bin-cells can be a problem when subscription regions are much larger than the bin-cell size (i.e., many bin-cells are contained within a subscription region) or much smaller than the bin-cell size (i.e., the filter will be inefficient because the resolution is too coarse). Federations should perform analysis of their filtering requirements to determine effective ranges and bin-cell sizes.

When a federate updates an object's attribute values, the RTI routes the update to the appropriate stream without knowing who the recipients are. The recipients are those federates that have subscription regions which overlap the corresponding update regions in the routing space bin-cells. All of this is transparently supported within the RTI.

## 5.3 RTI 1.1

Figure 7 and Figure 8 depict the major structural and data flow aspects of the RTI 1.1 design approach for Data Distribution Management[19]. A salient feature of these figures is a pair of managers: the Distribution Object Manager (DOM) and the Distribution Routing Manager (DRM). These managers encapsulate the algorithms and data structures specific to this design for DDM. The primary functions of the DRM, which is an RTI Core manager object, are routing calculations and connectivity establishment. The primary function of the DOM, which is a Generalized RTI Core Client (GRCC), is communication of routing information between Local RTI Components (LRCs). The intent is that the interfaces between the DOM/DRM and the surrounding manager objects should be abstract enough that different DDM implementations could be substituted without altering the interfaces.

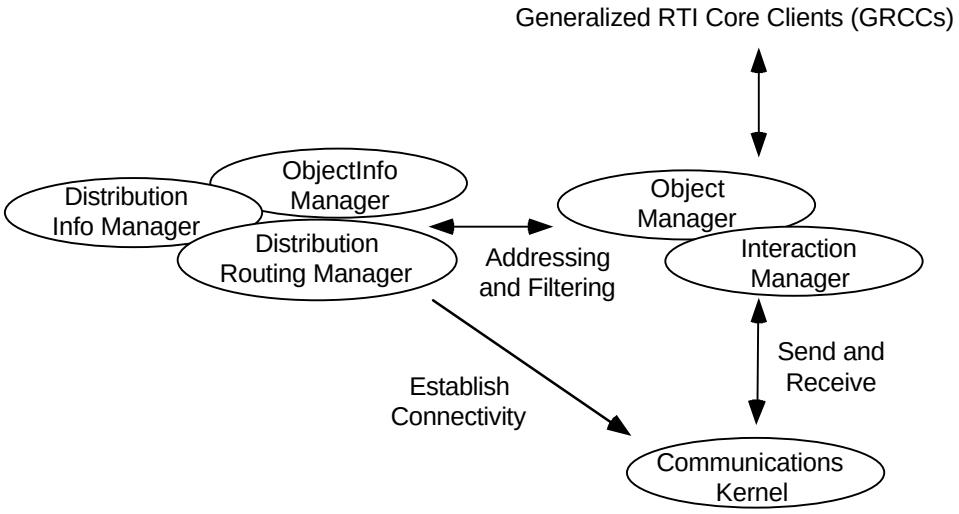



**Figure 7. Attribute/Interaction Forwarding Path**

[19] My thanks to Dan Van Hook for the RTI 1.1 Data Distribution Management Design Approach document from which this is excerpted.

The primary function of DDM is determination and control of routes for forwarding attribute updates and interactions. Figure 7 shows the forwarding path within the RTI, while Figure 8 illustrates the components involved in routing and their relationships. A key idea is that the RTI calculates routing information (e.g., addresses) and establishes connectivity outside of the path for forwarding data. When forwarding data, addressing information is looked up but not calculated. In Figure 7, the Object and Interaction Managers lookup addresses for sending updates and interactions using services of the DRM. Also, after receiving an update or interaction, the Object and Interaction Managers use DRM services to filter unsubscribed attributes or interactions that were delivered due to imprecise routing. A number of other managers, such as the Time Manager, are not shown.

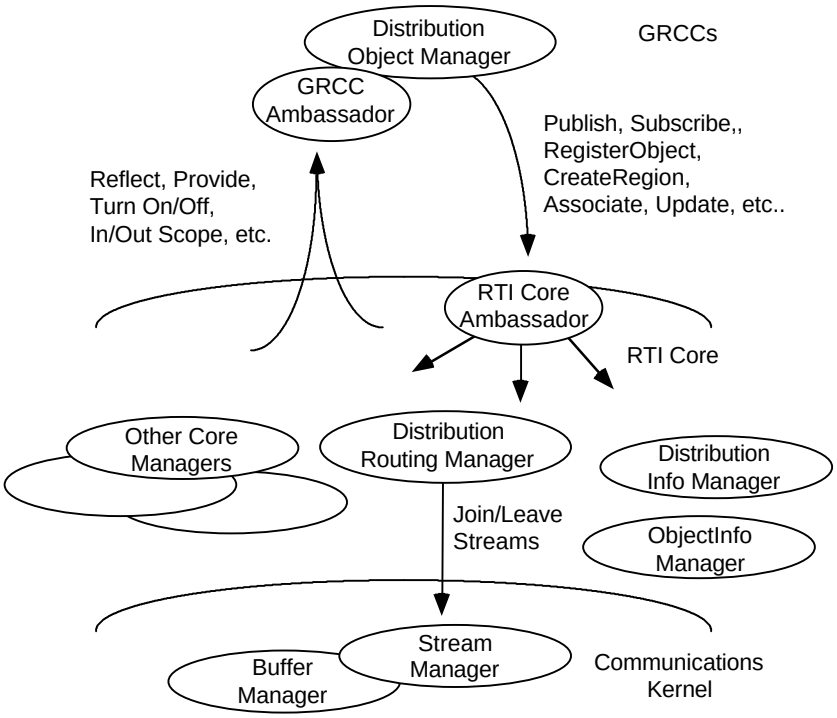



**Figure 8. Data Distribution Routing and Object Manager Interfaces**

### *5.3.1 Interfaces*

The interfaces shown in Figure 7 and Figure 8 may be grouped into three sets:.

- Communications Kernel
- RTI Core manager objects
- GRCC

Communications Kernel Interface**.** The DRM uses Stream Manager services to join and leave streams, depending on the routes it calculates.

RTI Core Manager Interfaces**.** The DRM has interfaces to other RTI Core manager objects. The DRM is an RTI Core manager and provides services such as:

- Region creation, modification, deletion, etc.
- Subscription for attributes and interactions with a region
- Subscription for attributes and interactions without a region (defaults are calculated)
- Association of attributes and interaction classes with regions
- Default region association for attributes of objects registered without a region
- Generation of Turn On/Turn Off, In Scope/Out Of Scope, and Provide services, e.g. to the Object Manager
- Modification of the contents of the Distribution Information Manager (DIM) database and Object Info Manager Database; these databases store the current collection of local and remote regions, associations, and subscriptions.
- Addressing lookup and filtering for sending/receiving data

GRCC Interface. The DOM is a GRCC. It invokes methods on the RTI Core through an RTI Core Ambassador. Similarly, the RTI Core invokes methods on the DOM GRCC through a GRCC Ambassador. These ambassadors are augmented with DDM-specific services supporting DOM/DRM interactions.

The DOM uses the RTI Core to communicate subscriptions and associations with peer DOMs residing in other LRCs. Subscriptions and associations are represented as DistributionObjects, which are instances of an RTI Core object class. DistributionObjects, their attributes, and data such as default order type, transport type, and routing space, are defined in the RTI Object Model (ROM), which is analogous to the FOM (Federation Object Model) but is RTI implementation dependent and private to the RTI.

### *5.3.2 Data Structures For Routing and Filtering*

Events that must be processed by the DRM, usually in response to activities of a GRCC (local or remote), include the following:

- Change extents for local region
- Change extents for remote region
- Change local attribute association
- Change remote attribute association
- Change local attribute subscription
- Change remote attribute subscription
- Change local interaction association
- Change remote interaction association
- Change local interaction subscription

- Change remote interaction subscription
- Determine relevance of received attribute update
- Determine relevance of received interaction class

For these events, a database of local/remote associations and subscriptions must be accessed to perform the following operations:

- Match local associations with remote subscriptions to determine subscriber sets for routing and to invoke Turn On/Turn Off services.
- Match remote associations with local subscriptions to invoke In Scope and Out Of Scope services.
- Determine subsets of attributes to be reflected and interactions to be received.

In addition, the Object Class and Interaction Class hierarchies must be traversed to support class promotion so that updates and interactions may be received as the subscribed class and not only as the registered class.

The initial DDM implementation will use a set of N-dimensional binary trees indexed by class (interaction or object), GRCC, and routing space. Each tree is a binary subdivision of a routing space. A one dimensional space results in a binary tree, a two dimensional space results in a quadtree, a three dimensional space results in an octtree, etc. Regions are inserted into each tree and region handles are stored at the nodes of the tree, depending on the portion of the space that a particular region overlaps. Trees have a maximum depth, specified in the RID.

These data structures will be maintained in the DRM as components of routing space class instances. Regions, associations, subscriptions, (both local and remote) will be maintained in the DIM.

### *5.3.3 Mapping to Streams and Establishing Connectivity*

The matching operation results in a set of subscribers for each region. The associated data – attributes or interactions – must be delivered to those subscribers. To make this possible, the subscriber set must be mapped to a stream on which the associated data will be sent. In addition, the subscribers must be joined to that stream.

Mapping from subscriber sets to streams is accomplished by lookup in a static group mapping table loaded at initialization. The stream on which data associated with each region is sent is stored with that region and looked up by RegionHandle when data is sent. In the initial implementation, subscribers join all relevant streams based on the contents of the group mapping table. Later implementations may permit dynamic connectivity as well as dynamically recalculated group mapping tables.

## 6. Bibliography

James O. Calvin, Carol J. Chiang, Stephen M. McGarrry, Steve J. Rak, Daniel J. Van Hook. 1997. “Design, Implementation, and Performance of the STOW RTI Prototype

(RTI-s)." In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, 97S-SIW-019.

Cisco Connection, 1996. "Documentation." Available through the Internet: *http://www.cisco.com/univ-src/3.7/home/home.htm*. Vol. 3, No. 7.

Deering S. 1989. "Host Extensions for IP Multicasting.", *Network Working Group, rfc1112*.

DMSO 1996. "HLA OMT Extensions Version 1.0, August 20, 1996." Available through the Internet: *http://www.dmso.mil/hla*/.

DMSO 1997. "HLA Frequently Asked Questions." Available through the Internet: *http://www.dmso.mil/hla.*

DMSO 1997. "HLA Interface Specification Version 1.2, August 1, 1997." Available through the Internet: *http://www.dmso.mil/hla*/.

DMSO 1997. "HLA Rules Version 1.2, August 13, 1997." Available through the Internet: *http://www.dmso.mil/hla*/.

DMSO 1997. "HLA Object Model Template (OMT) Version 1.2, August 13, 1997." Available through the Internet: *http://www.dmso.mil/hla*/.

DMSO 1997. "HLA Time Management: Design Document Version 1.0, August 15, 1997." Available through the Internet: *http://www.dmso.mil/hla*/.

Fujimoto R. 1990. "Parallel Discrete Event Simulation.", Communications of the ACM. Vol. 33, No. 10, Pages 30-53.

Mellon L. 1996. "Hierarchical Filtering in the STOW System." In *Proceedings of the 15th DIS Workshop*.

Morse K. 1997. "Differentiating Declaration Management and Data Distribution Management." In *Proceedings of the 1997 Fall Simulation Interoperability Workshop*.

Morse K. 1996. "Interest Management in Large Scale Distribute Simulations." *University of California, Irvine, Technical Report TR 96-27*.

Petty M. 1997. "Experimental Comparison of d-Rectangle Intersection Algorithms Applied to HLA Data Distribution." In *Proceedings of the 1997 Distributed Simulation Symposium*.

Seidensticker S. 1996. "HLA Data Filtering/Distribution Requirements.", In *Proceedings of the 15th DIS Workshop*.

Steinman J., Wieland F. 1994. “Parallel Proximity Detection and the Distribution List Algorithm.” In *Proceeding of the 8th Workshop on Parallel and Distributed Simulation (PADS94)*. Pages 3-11.

Symington S., Weatherly R., Little R., Calvin J.. 1997. “A State Transition View of Updates and Interactions.” In *Proceedings of the 1997 Fall Simulation Interoperability Workshop*.

Van Hook D. 1997. “RTI 1.1 Data Distribution Management (DDM) Design Approach Version 0.2, October 17, 1997.”

Van Hook D., et. al. 1996. “Performance of STOW RITN Application Control Techniques.”

Van Hook D., Rak S., Calvin J. 1996. “Approaches to RTI Implementation of HLA Data Distribution Management Services.”, In *Proceedings of the 15th DIS Workshop*.

Van Hook D., McGarry S. 1996. “A Prototype Approach to the Data Communications Component of the RTI.”, In *Proceedings of the 15th DIS Workshop*, (previously titled, “An Update on the RTI Design”).

Whetten B., Montgomery T., and Kaplan S. 1994. “A High Performance Totally Ordered Multicast Protocol.” *Theory and Practice in Distributed Systems*, Springer Verlag LCNS 938.